

Monte Carlo Transport Calculations on an Ultracomputer

Y-Q. Zhong

M.H. Kalos

CONTENTS

- I. INTRODUCTION
- II. PROBLEMS AND RESULTS
- III. EFFICIENCY AND RELATED ISSUES
- IV. PSEUDO-RANDOM GENERATORS

APPENDICES

- A1. AN IDEALIZED MODEL OF STRAGGLING
- A2. CONGESTION IN BATCH STORAGE
- A3. NON-OVERLAPPING OF RANDOM NUMBER SEQUENCES

Acknowledgements

References

Tables

I. INTRODUCTION

We report the results of some experiments in adapting some (existing) Monte Carlo transport codes to a simulated paracomputer. A paracomputer is an idealized model of an array of parallel processors (PE's) in which each processor can follow its own instruction stream and all can access any address in a shared memory in one cycle [1]. While such machines cannot be built, the NYU Ultracomputer is designed to be a close approximation. Computers of this kind are said to have MIMD (Multiple Instruction Stream, Multiple Data Stream) organization. It would certainly seem plausible that Monte Carlo transport codes could be readily and efficiently adapted to this kind of architecture. We find that to be true and explore some subsidiary technical issues pertaining to efficiency and to details of the program transformation.

Adaptation of transport Monte Carlo to SIMD (Single Instruction Multiple Data) machines such as the ILLIAC IV [2] or contemporary vector machines has been studied [3]. Although it turns out to be possible, the code transformation is very complicated. It is not an exaggeration to say that the codes must be completely restructured at a very deep level. Also certain techniques commonly used (e.g. rejection techniques) are difficult to incorporate so that parts of the algorithm must be rewritten.

We expect - and find - that only a relatively small effort is required to restructure for an MIMD architecture a code written for a serial machine. The shared memory is one feature that makes the Ultracomputer especially easy to use for large Monte Carlo calculations. In particular, if we assume that it is important to treat large cross sections sets, large geometrical descriptions, and to obtain a large set of fluxes and their errors, then it is extremely convenient and efficient to be able to reference any datum without having to worry about its location in a hierarchical memory.

In a practical Ultracomputer, access to shared memory is expected to be somewhat slower than in contemporary serial machines. For that reason, and to diminish traffic on the network that connects processors to shared memory, it is planned to add local memory (organized as a cache) to each processor. Separately addressable local memory is not

envisioned for the system at the present time. The simulator [4],[5] used supports the distinction between "private" and "shared" memory. The former is used for values needed by one processor, the latter for those to which many processors need to refer. We assume that private variables would be cached. In addition, as we discuss presently, Monte Carlo codes usually have a large read-only shared data base which may also be cached.

In adapting a code for the simulator, it is necessary to make a choice of what computer variables are private or shared. We assumed that the calculations would normally be carried out on a problem with a very large data base of cross sections and geometry and therefore assigned those to shared memory. In fact, this was not true of the problem actually simulated so we gave up some potential efficiency in the interest of making a simulation of more general interest. We also assumed that many answers would be tallied. Because of this, and because all processors contribute, the arrays of answers are put in shared memory. Clearly, variables used only for generating different histories can and should be kept in local memory. That means that in the simulator, they are declared as private. Since cross section and geometrical data (along with instructions) are cacheable without restriction (i.e., are "read only"), most of the references to shared data can be effectively treated.

Updating particle tallies in a large number of different spatial, energy, time, and direction bins by different processors presents a serious problem in some machines. This difficulty is well resolved by a new instruction which was devised for coordination. It is the "fetch and add" which indivisibly adds an increment to memory. We assume the availability of a floating point fetch and add.

The efficiency of a parallel program is defined as

$$\text{eff} = \frac{T_1}{n \cdot T_n},$$

where T_1 is the time needed using one processor and T_n is the execution time when the same problem is run by n processors. When the algorithm

is changed from serial to parallel, T_1 is somewhat larger than in the serial case, since we include the overhead of the parallel code even for one processor. Generally this efficiency will go down when the number of processors increases and the problem complexity is fixed since a larger fraction of time is (usually) devoted to coordination. In addition the difficulty of keeping all processors busy (i.e. of balancing the computation load among the processors) may become more difficult. Some attention to the design of a parallel algorithm may be required to keep the efficiency high. The obverse of the general decline of efficiency with processor number is that as the problem size grows (here the total number of histories), the number of processors that can effectively be kept busy also grows. Our simulation results will illustrate these points. In what follows, we discuss some of the qualitative issues that affect the efficiency of parallel Monte Carlo transport programs.

In any Monte Carlo calculation it is highly desirable to obtain an estimate of the standard deviation of each answer. A useful (though not always optimal) way of doing this is to divide the total history count into fairly large batches, say M histories per batch. Then the mean and mean square value per batch of each answer is obtained. This presents a problem for an Ultracomputer when the total number of bins for all statistical batches cannot be kept in memory. When M histories are completed, the squares of each tally can be computed, but the history generation process must be stopped lest the tallies be changed by histories in the next batch. The natural fluctuations in time means that the computer is insufficiently used between the time that the M^{th} history is begun and the time when the last history of a batch terminates. This straggling effect can be reduced by assigning new histories to processors as they become idle rather than preassigning a fixed number to each. To make the idle time still smaller we used duplicate arrays for odd and even numbered batches. The effect of this and of other straggling effects is discussed and analyzed. The dispersion in computing times is an important effect. Even with use of two arrays it is necessary to use semaphores to protect the arrays against entry during histories in later batches.

The calculation actually carried out included two fixed source

problems and two in which eigenvalues are estimated by means of power iterations. We are confident that the results as far as Ultracomputer efficiency is concerned are representative of a wide range of transport Monte Carlo. Issues affecting this will be discussed.

In Section II the physics and the Monte Carlo algorithms of the four problems solved are briefly discussed. The computational results are given in the tables. The transformation from serial to parallel code is also described. In Section III we discussed the efficiency of parallelization, the efficiency of a Monte Carlo algorithm on a paracomputer and the issue of references to private and shared data. Besides these, the choice of random number generators to use in an Ultracomputer is an important issue. This is the topic of Section IV.

II. PROBLEMS AND RESULTS

Four neutron-transport problems are solved with Monte Carlo methods, including both fixed source and eigenvalue problems.

Mathematically, the fixed source problems correspond to inhomogeneous integral (or integro-differential) equations, while the eigenvalue problems seek the fundamental eigenvalue and eigenfunction of an integral (or integro-differential) equation. For fixed source calculations, the distributions of initial particle positions, directions and energies are specified. On the other hand, in eigenvalue problems these distributions are unknown. Trial distributions are assumed and iterated until the equilibrium distribution is established. Then the calculation becomes similar to a fixed source problem. A practical issue in eigenvalue problems is the reduction by some acceleration scheme of the number of iterations required to achieve a needed accuracy.

In adapting a serial code to a parallel version, the fetch and add instruction was extensively used to sum Monte Carlo estimates of all estimators, and semaphore techniques [6] are used to control concurrent execution.

Necessary changes for revising a serial code to a parallel code comprise the following:

A. Define the physics and geometrical data, the tally storage, and

the necessary semaphore variables as shared data; all other data, particularly those which pertain to individual histories such as the position, direction, track length of the particles, are defined to be private.

B. Change the DO loops for histories and batches into semaphore-related constructions, including the initialization, their fetch-and-add revision, and the test for completion. Usually two sets of storage are maintained to score two batches alternately.

C. All scores are summed using real fetch-and-add to facilitate concurrent execution.

The following is a description of the problems:

Problem 1.

Calculate the ratio of the flux at a point to the average flux on a circle containing the point. The adjoint mode in which radiation is followed from detector to source was employed, along with correlated sampling to reduce the variance of the ratio. A history is divided into three stages. The first stage belongs to both the point and the circle, while the second one is associated with flux at the point and the third one is for the flux on the circle.

In the original (sequential) version of the program, a process is defined as the whole history from Stage 1 through Stage 3. Results of parallel computation on this problem are shown in Table 1A.

The correlated sampling procedure included using four randomly sampled initial positions on the circle; the procedure is similar to that of splitting. As a refinement, a new version takes that part of the history in Stage 1 and 2, or the part in Stage 3 for one position as one process. At the end of Stage 1 four more processes are created and stored in a queue to be available for processing. From Table 1B we can see the efficiency is improved. This is anticipated since two effects improve the efficiency, one being the increased number of processes which can be carried out in parallel, and the other being the decreased fluctuation in the length of the computation. This kind of improvement has been carried out earlier in [7].

Problem 2.

The physical problem here is the same as in Problem 1. Now we use the adjoint mode with a first common collision position, using weights to score the point flux and the circle flux and introducing an optimized first collision density distribution. The simulated results are given in Table 2.

Problem 3.

This is a one-dimensional diffusion problem, seeking the fundamental eigenvalue and its corresponding eigenfunction. Here we did not carry out a detailed simulation of particle tracks in the usual way. Instead Green's function for the time independent diffusion operator with constant absorption was used to sample next fissions

given a previous one. This gives a simplified problem in which we could explore the issues related to power iteration and the fluctuation in the time to complete the generations. In reprogramming for the parallel case, we have two options: one is to store in memory all the scores for every generation, which is likely to be impractical when many different quantities are computed over many generations; another is to keep only limited storage for the histories. In particular one could reserve enough shared memory for all flux estimates during one generation. But it could happen that some processors would finish their histories for that generation while others were still working on their own. In that case the program has to wait until the tally space is released. This semaphoring will lower the efficiency somewhat, because the processors finishing earliest idle until the slowest is ready. Results are shown in Table 3A and 3B.

Problem 4.

In order to get a clearer sense concerning reactivity calculations, a bare homogeneous sphere was chosen as a model to study. In Table 4 we give the computational results.

III. EFFICIENCY AND RELATED ISSUES

A. In Appendix A, the loss of efficiency because of history straggling is analyzed. Using an idealized model of the fluctuations in execution time, we can obtain an approximation for the efficiency: [A1]

$$\text{eff} = \frac{1}{1 + \frac{m}{N}(\log m + \gamma - 1)},$$

where m is the number of processors, N the number of histories, and γ is Euler's constant, 0.5772... Table 5 gives some results from this formula. For practical cases, we must also consider the effect of the limited storage available for flux tallies. Usually two areas for flux tallies are maintained to reduce the wait until such areas become free,

as described in I. Then there would be some chance for the histories in the third batch to have to wait until the first or second batch is completed. Under mild assumptions, the probability for this event to occur is [A2]

$$p \approx 1 - e^{-\lambda}, \text{ with} \\ \lambda = m \cdot e^{-k/m},$$

where m is the number of processors, k is the number of histories per batch, and λ is the expected waiting time in units of the mean time for one history. Table 6 shows the expected waiting time for some m and k .

Combining the two effects mentioned above, we can calculate the efficiency for various number of processors, as Korn [8] did for some algebraic problems.

A Monte Carlo simulation was carried out to do this calculation. Given N PE's and M histories, and a random variable, t , representing the computer time for processing a history, we can simulate the efficiency of parallel execution.

Three alternatives were used for the distribution of the random time mentioned above: The exponential distribution, i.e.

$$p(t) = e^{-t}, \quad t > 0 \quad (1)$$

or the geometrical distribution, i.e.

$$p(t=n) = p_s^{n-1} p_c, \quad (n = 1, 2, \dots) \quad (11)$$

where $p_c + p_s = 1$, p_c is the capture probability and p_s is the survival probability in one collision, or a composite of (1) and (11), viz.,

$$t = \sum_{j=1}^n t_j, \quad (111)$$

with each t_j distributed as e^{-t} , and n following the law given in (11). Usually, this case is closest to reality.

Table 7-A gives the estimated efficiency for the above three distributions and for the preassigned equal number of summands with

various numbers of PE's (denoted as N) and total histories $M = 128$ or 1024. As a Monte Carlo calculation, 20 experiments were performed for each situation, and the average (the ratio of the total sums, not the mean value of the ratio) is taken as the estimate. There is no significant difference among the efficiencies corresponding to the three different distributions. On the other hand, the case of equal number of summands shows a distinct drop in efficiency in comparison with the more practical three cases.

In Table 7-B the data are shown for the case of the composite distribution and the practical parallel calculation. It is observed that for the 100 history case the practical results reached a higher efficiency than the idealized one. However when the observation continues, as the number of histories becomes 300, the situation reverses, especially for large a number of PE's. Consequently the model was amended to incorporate the use of two sets of storage for scoring as many as two batches concurrently and the probable waiting time was simulated. It is very interesting to see that after this improvement in the model, the efficiency for #PE = 32 in the idealized case becomes 0.738, a very close approximation to the real value of 0.734.

It is hoped that this model can be used to predict efficiency on an Ultracomputer. The computation of the model is much less expensive. However, it should be noted that if the times spent on different processes are not independent of each other, or are not identically distributed, this model must be modified to fit the true situation. It is not clear that this would ever be necessary.

Table 7-C lists the cases for three different values of P_s as required in distributions (ii) and (iii).

Consideration should be given to the fact that after completing each batch there is an editing phase, which contributes to the delay. (See Table 3 and 4).

B. On a serial computer the efficiency ϵ_s or the quality of a Monte Carlo method is defined as inversely proportional to the product

of the variance (Var_0) of the estimator and the time spent in obtaining one estimator, t_0 . That is

$$\epsilon_s \propto 1/(\text{Var}_0 * t_0) .$$

In fact, $t_0 = E(t)$ is the expectation of t since the time t spent for one estimator is itself a random variable.

It is natural to generalize this definition to the paracomputer by

$$\epsilon \propto \text{eff}/(E(t) * \text{Var}_0) ,$$

where eff is the efficiency of parallel execution as defined before.

This relation calls one's attention to the fact that in choosing a particular Monte Carlo scheme run on a paracomputer one should consider both the variance of the estimator and of the time spent for each process, which affects the efficiency of parallel execution.

C. The ratio of references to shared data compared to references to private variables.

Such ratios in problem 3 and 4 are much higher than those in problem 1 and 2; however, this is not inherent. There should be some increase in the ratio for Problem 3 and 4, since the secondaries are stored in the common pool, but in the above problems the main fact is that the history is short and the geometry is simple. When the geometry is complicated, tracking will produce many intermediate results, especially for example, the solutions of the quadratic equations in searching for the crossing boundary. Of course these results will be kept in private variables (to be cached in a practical machine).

D. It is typical in the eigenvalue problems that the number of histories per generation is not fixed, though in the case of fixed source we would also meet with this issue if a splitting technique were adopted. We can take some precaution to prevent the population from overflowing or underflowing. After one generation is completed some indication can be obtained from the eigenvalue estimate or the

population in this generation. An adjustment can then be made to the trial eigenvalue in succeeding generations. However, a compromise should be made either to perform the adjustment without waiting for the end of all current batches and tolerate a discrepancy among the results when the problem is run using different numbers of processors, or to make each edit a critical section and suffer a loss of efficiency, or to delay the adjustment to the untouched generations and run the risk of losing control of population.

In practice, this presents no serious trouble. We can run the problem in advance and acquire enough necessary information about the magnitude of eigenvalue and population. Then the problem will be run using these results to control the population without depending on the details of the current generation's population.

E. As pointed out in [3], rejection techniques in sampling random variables are disadvantageous on vector architectures. This no longer holds on an MIMD machine. Still, rejection methods have variable computing times which is undesirable from the point of view of efficiency if the variation is really very large. An example is examined, in which a point is to be chosen uniformly on the unit circle. The direct scheme is that

$$x = \cos(2\pi\xi)$$

$$y = \sin(2\pi\xi)$$

with ξ sampled as random number. On the other hand, the rejection method is the well-known von Neumann device: i.e., first the point is picked on the unit square and accepted if in the enclosed circle. Table 8 shows the result of numerical experiments. In this case, the rejection method is still preferable to the direct one. Thus the MIMD architecture permits the use of techniques rather unfavorable for vector machines [3].

IV. PSEUDO-RANDOM GENERATORS

The conventional generator uses the homogeneous multiplicative congruential method. If the number of significant digits is p , and the modulus is taken as 2^p , the period of this generator will be 2^{p-2} . For

an Ultracomputer we are particularly concerned with very large calculations and so expect to consume a great many random numbers. In addition we would like to assign to each process into which a calculation is subdivided its own local generator so as to avoid the serial bottleneck associated with a global generator. Another, more subtle and more serious problem with a global generator is that it may lead to the result that the outcome of a Monte Carlo calculation depends upon the order in which processes are initiated and hence upon the operating environment. In using local generators we must, of course, ensure that the probability is very small that two processes use the same sequence, even in part. A long period will help. Tausworthe generators can supply us with a sufficiently long period, but its drawback for the Ultracomputer is that initialization is likely to be slow and this will have to be carried out even for short processes. In addition each processor will require a few hundred locations saved for the generator.

An ideal scheme could work in this way: for each history or process the multiplicative congruential generator produces the sequence of random numbers while the seeds are generated by another generator. In fact in the work described here, each process seeded its successors, as will be described below. This is the easiest and, given a good way of generating a seed from the last random number of a process, a robust method. It provides the essential feature mentioned above that a Monte Carlo simulation be repeatable in exact detail.

It is shown in (A3) that the non-overlapping probability p_0 of the sequences produced with this scheme depends on the length τ of the period of the congruential generator as well as the number m of the random numbers used in one history and the total numbers n of histories followed. We have the result that

$$p_0 = (1 - \frac{m}{\tau})^{n-1} .$$

A problem run on an Ultracomputer might follow as many as 10^6 histories. If the period is 2^{48-2} as in the case on CDC 6600, the above scheme is not quite good enough. But if the period is 2^{72} or

even 2^{64} , this device can work fairly well. (cf. Table 9, some examples of P_0).

In our experimental calculation, each history used a sequence of random numbers produced by the congruential method. The first entry of this sequence (or the seed) is generated beforehand if the history belongs to the first generation, and all the seeds for first generation are known to be far apart from each other in the congruentially generated sequence. The seed of the history in the succeeding generations is produced by the mid-square method using the parent history's last random number as its predecessor. If there are several secondaries for one history, the mid-square method is applied as many times as necessary to supply the seeds for secondaries. This scheme is different from the purely mid-square approach, which justifiably has a bad reputation. An alternative to the mid-square that we used is simply a different multiplicative generator for generating the seeds for the successor processes.

APPENDICES

A1. AN IDEALIZED MODEL OF STRAGGLING

Assume there are m procesors and N histories to be tracked, and the time spent on following a history is distributed as the exponential one, i.e. its p.d.f.

$$p(t) = \lambda \cdot e^{-\lambda t}, \quad t > 0.$$

As soon as one history is finished, another begins on that processor unless the N 'th history has begun. By any time t , the system might have already completed n histories, $n = 0, 1, 2, \dots$. The probability of finishing n histories is

$$P_n(t) = \frac{(\mu t)^n}{n!} e^{-\mu t}, \quad n = 0, 1, 2, \dots \quad \text{A1-1)}$$

where

$$\mu = m\lambda. \quad \text{(A1-2)}$$

Suppose for any fixed value of n , at exactly the time t the system has just finished the n 'th history. The probability that the above event happened in the interval $(t, t+dt)$ is

$$f_n(t)dt = P_{n-1}(t)\mu dt \quad n = 1, 2, \dots \quad \text{(A1-3)}$$

while $f_0(t) = \delta(t)$. It is easy to obtain the expectation of t for fixed n

$$E(t; n) = \frac{n}{\mu} = \frac{n}{m} \cdot \frac{1}{\lambda}, \quad \text{(A1-4)}$$

Now the time t the system spends to finish the whole job can be divided into two parts t_1 and t_2 , $t = t_1 + t_2$. The time t_1 is defined as the time spent on the first $(N-m)$ histories, whose distribution is obtained from Eq. (A1-3), with $n = N-m$. On the other hand after time t_1 any of all the N histories either has been completed or at least has begun; in fact the last m unfinished histories are loaded one per PE. When one of these PE's finishes its history, it becomes idle until the last history finishes; when all have finished the system completes its current job. The interval from t_1 to the end is defined as t_2 .

According to our assumptions about the computing time distribution, t_2 is distributed as the extreme of a sample with volume m . For the exponential p.d.f.,

$$E(t_2) = E(x_{(m)}) = \sum_{i=1}^m \frac{1}{i} \times \frac{1}{\lambda}$$

$$= (\log m + \gamma)/\lambda, \quad (A1-5)$$

where $\gamma = 0.5772\dots$ is Euler's constant [9].

Hence we have

$$E(t) = E(t_1) + E(t_2)$$

$$= \left(\frac{N-m}{m} + \log m + \gamma\right)/\lambda, \quad (A1-6)$$

So the efficiency of the parallel execution is

$$\text{eff} = \frac{N \cdot \frac{1}{\lambda}}{m \cdot E(t)} = \frac{1}{1 + \frac{m}{N}(\log m + \gamma - 1)}. \quad (A1-7)$$

A2. CONGESTION IN BATCH STORAGE

A2.1. Statement of the problem

There are m PEs, each able to process one history at a time. When a history is completed, a new history is begun. The histories are numbered according to their appearance in the system as a whole.

There are two flux tally areas, each serving scores of one batch. Every batch will include q histories. The first to q^{th} histories are scored in Batch 1; while the $(q+1)^{\text{st}}$ begins, the second batch storage is filled.

If the $(2q+1)^{\text{st}}$ history is to begin and neither of the previous batches has been released, we say that congestion occurs. The probability of this event is to be evaluated, as well as the expected waiting time for the storage to be freed.

2. The assumptions

It is reasonable to assume that

(a) The computing time for a history is independent of the other histories and of the processors.

(b) Markovian property: Provided that at this moment the particle is alive, it will live a time beyond this instant which is independent of how long it has lasted so far.

Under these assumptions, the probability of termination of a history on each PE at any given moment is $1/m$, given the condition that among all the PEs there is exactly one which finishes a history.

3. The derivation of the formulae.

In fact this is a classic problem in probability.

We denote by A_i the event that after the $(q+1)^{\text{st}}$ history begins on a PE, the i^{th} PE will not begin a history which is numbered from $(q+2)$ to $(2q+1)$. The index i should not be the same one as that on which the $(q+1)^{\text{st}}$ history takes place. (For convenience, it is numbered as m^{th} PE).

The event of failure F is equivalent to

$$F = \bigcup_{i=1}^{m-1} A_i .$$

The probability of this event is

$$\begin{aligned} P\{F\} &= \sum_{i=1}^{m-1} P\{A_i\} - \sum_{i < j} P\{A_i \cap A_j\} + \sum_{i < j < k} P\{A_i \cap A_j \cap A_k\} \\ &\quad - \dots + (-1)^{m-2} P\left\{ \bigcap_{i=1}^{m-1} A_i \right\} \\ &= \sum_{k=1}^{m-1} (-1)^{k-1} \binom{m-1}{k} \left(1 - \frac{k}{m}\right)^q \end{aligned} \quad (A2-1)$$

It should be noted that this definition of F has included a situation which turns to be a success, that is when all q histories from $(q+2)$ to $(2q+1)$ appear on the m 'th PE. The probability of this case is just $(1/m)^q$, normally negligible.

In all practical cases, the values of m and q are too large to use the formula (A2-1). There exists an approximation to Eq. (A2-1), for $m \gg 1$ and $q \gg 1$, when $q/m \gg 1$ is bounded above,

$$P\{F\} \doteq 1 - e^{-\lambda} , \quad (A2-2)$$

with

$$\lambda = me^{-q/m} . \quad (A2-3)$$

The more practical question is how long must the system wait until storage is available. For this purpose, the probability that exactly k ($k = 1, 2, \dots, m-2$) PEs will not finish their histories during the sequence of termination of q histories must be evaluated. As in the derivation of (A2-1), these probabilities are given as

$$P_k \doteq \lambda^k e^{-\lambda} / k! \quad k = 1, 2, \dots \quad (A2-4)$$

And the expected waiting time is thus

$$E(\tau) = \lambda , \quad (A2-5)$$

The unit here is the average computing time for one history.

Note: If there are three areas for scoring three batches at the same time, the probability of a failure will drop substantially.

A3. NON-OVERLAPPING OF RANDOM NUMBER SEQUENCES

If a method is applied to generate a great many seeds, each of which is used to produce a sequence of random numbers by a congruential generator, then it should be investigated whether these sequences of random numbers can be considered independent. It is assumed that the multiplier of the congruential generators yields satisfactorily independent pseudorandom numbers and that non-overlapping subsequences are also independent.

Suppose the length of each sequence is at most m , the period of this congruential generator is τ , the possible maximum number of sequences (or seeds, or histories in a Monte Carlo calculation) is n . The problem becomes as follows: what is the probability that within all the n sequences, or all the $n \times m$ entries, there is no overlapping at all. This can be changed into a geometrical problem:

Pick n points on a circle with circumference 1 and find the probability that all these n points are separated from each other by a distance of at least δ , $= m/\tau$.

It can be shown that this probability is

$$P_0 = (1 - n\delta)^{n-1} . \quad (A3-1)$$

We also can prove that the probability of the event that there are exactly k points such that the distance between any of them and its clockwise next point is smaller than δ is

$$P_k = \binom{n}{k} \Delta_k (1-n\delta)^{n-1}, \quad k = 0, 1, \dots, n-1, \quad (A3-2)$$

where Δ_k is the difference operator, which means

$$\Delta_k (1-n\delta)^{n-1} = \sum_{j=0}^k (-1)^j \binom{k}{j} (1-(n-k+j)\delta)^{n-1}. \quad (A3-3)$$

Specifically,

$$P_0 = (1-n\delta)^{n-1}$$

$$P_1 = n[(1-(n-1)\delta)^{n-1} - (1-n\delta)^{n-1}]$$

.

.

.

(A3-4)

$$P_{n-1} = \binom{n}{n-1} \Delta_{n-1} (1-n\delta)^{n-1} = n! \delta^{n-1}.$$

Proof. Choose an arbitrary point on the circle as the origin, and order the n points on the circle in clockwise fashion. Thus $0 < x_1 < \dots < x_n < 1$. We can always take x_1 as the origin. Let $y_1 = x_1 - x_1$, $i = 2, \dots, n$. Then we have

$$0 < y_2 < \dots < y_n < 1.$$

This region in $(n-1)$ -dimensional space constitutes all possible points, and its volume is easily seen to be

$$1/(n-1)!$$

The probability P_0 corresponds to the set of all points

$$\delta < y_2$$

$$y_2 + \delta < y_3$$

...

$$y_{n-1} + \delta < y_n < 1-\delta$$

Integration shows that the volume of this region is

$$(1-n\delta)^{n-1}/(n-1)!$$

Hence

$$P_0 = (1-n\delta)^{n-1}.$$

In order to prove the formula for P_k , we notice that the factor $\binom{n}{k}$ counts the number of ways to choose k points among n points. When the k points have been chosen, write them as $(x_{i_1}, \dots, x_{i_k})$, which satisfy

$$x_{i_j+1} < x_{i_j} + \delta, \quad j = 1, 2, \dots, k.$$

For this set of points $(x_{i_1}, \dots, x_{i_k})$, the region is decided by

$$x_\ell < x_{\ell+1} < x_\ell + \delta, \quad \text{if } \ell = i_j.$$

or

$$x_\ell + \delta < x_{\ell+1}, \quad \text{if } \ell \neq i_1, \dots, i_k$$

Transforming by $v_\ell = x_{\ell+1} - x_\ell$ if $\ell = i_j$ or $v_\ell = x_{\ell+1} - x_\ell - \delta$ if $\ell \neq i_1, \dots, i_k$. We obtain

$$0 < v_l < \delta, \quad \text{if } l = i_j,$$

$$0 < v_l, \quad \text{if } l \neq i_1, \dots, i_k.$$

$$\sum_{i=1}^{n-1} v_i = x_n - x_1 - (n-k-1)\delta < 1 - (n-k)\delta.$$

Now we see that the volume of the region is unrelated to the special choice of $\{i_j\}$. If we choose to compute the volume of the region

$$0 < v_i < \delta, \quad 1 \leq i \leq k$$

$$0 < v_i, \quad k < i \leq n-1$$

$$\sum_{i=1}^{n-1} v_i < 1 - (n-k)\delta.$$

then

$$\begin{aligned} P_k &= \binom{n}{k} \int_0^\delta \cdots \int_0^\delta (1 - (n-k)\delta - \sum_{i=1}^k v_i)^{n-1-k} dv_1 \dots dv_k \times \frac{(n-1)!}{(n-1-k)!} \\ &= \binom{n}{k} \Delta_k (1 - n\delta)^{n-1}. \end{aligned}$$

The evaluation for P_k and

$$E(k) = \sum_k k P_k$$

is difficult to perform by the above formulas. Fortunately, there is an approximation to it. Assume

$$n\delta \ll 1.$$

and $n^2\delta$ is a bounded value. Then we have

$$P_k \doteq \frac{\mu^k}{k!} e^{-\mu}, \quad k = 0, 1, 2, \dots \quad (\text{A3-5})$$

where $\mu = n^2\delta$. Therefore

$$E(k) \doteq \mu. \quad (\text{A3-6})$$

But we still can use the formula

$$P_0 = (1-n\delta)^{n-1}$$

to calculate the probability of non-overlapping. To demand that all the entries be different is a strong restraint. A reasonable criterion is to have a small number of overlaps, i.e. $E(k)$ is small. It should be pointed out that overlap does not mean the histories coincide with each other, since the random number sequences in question may still play different roles in the Monte Carlo calculations.

Acknowledgements

One of the authors (Y.Z.) would like to express his gratitude to A. Gottlieb and B. Lubachevsky for their help and suggestions in programming the simulations. This work was supported in part by the Applied Mathematical Sciences program under DOE Contract DE-AC02-76ER03077, and in part by the National Science Foundation under Grant No. NSF-MCS 79-07804.

References

- [1] J.T. Schwartz, "Ultracomputers", ACM TOPLAS, Vol. 2, No. 4, pp. 484-521, Oct. 1980.
- [2] E. Troubetzkoy, H. Steinberg, M. Kalos. Monte Carlo Radiation Penetration Calculations on a Parallel Computer, Trans. A. Nucl. Soc., 17, 260 (1973).
- [3] F.B. Brown, Vectorized Monte Carlo, (doctoral thesis) The University of Michigan, Aug. 1981; Vectorized Monte Carlo Methods for Reactor Analysis, Proceedings of the ANS Topical Conference "Advances in Reactor Computations", Salt Lake City, March 28-31, 1983. ISBN 0-89448-111-8
- [4] A. Gottlieb. Washcloth - The logical successor to Soapsuds, (Ultracomputer Note #12), Dec. 1980.
- [5] A. Gottlieb. Washcloth 81 (Ultracomputer Note #21), Jan. 1981.
- [6] A. Gottlieb et al. Basic Techniques for the Efficient Coordination of Very Large Numbers of Cooperating Sequential Processors. (Ultracomputer Note #16), Dec. 1980.
- [7] M.H. Kalos, Scientific Computations on the Ultracomputer, (Ultracomputer Note #30). Sept. 1981.
- [8] D. Korn, Timing Analysis for Scientific Codes Run under Washcloth Simulation. (Ultracomputer Note #24) Mar. 1981.
- [9] E. J. Gumbel, Statistics of Extremes, p. 90, Columbia University Press, New York 1958.

Tables

#PE	Time	eff	#PE	Time	eff
1	17014	1.	1	15973	1.
4	4309	0.987	4	4028	0.991
8	2252	0.944	8	2052	0.973
16	1239	0.858	16	1061	0.941
32	724	0.734	32	564	0.885

References to

Shared	216	Shared	232
Private	5146	Private	4895
S/L(%)	4.2	S/L(%)	4.7

Table 1A. Problem 1, one history-
one process.

Table 1B. Problem 1, split
history as one process.

Note: Three batches, each batch being 100 histories. "Time" is the number of instructions performed, in thousands. The number of references is obtained with 16 PEs, also in thousands. (Except for the number of histories, this remark holds for the other tables below.)

**The efficiencies in Table 1B are higher than those in Table 1A. Although the Shared/Private ratio is also increased, an unfavorable change, the difference is insignificant.

#PE	Time	eff	References to
1	8070	1.	Shared 122
2	2052	0.983	Private 2528
8	1046	0.964	S/L(%) 4.8
16	559	0.902	
32	316	0.798	

Table 2. The results in Problem 2.

#PE	Time	eff	#PE	Time	eff
1	4965	1.	1	5129	1.
8	633	0.980	8	642	0.999
16	326	0.952	16	327	0.980
32	172	0.903	32	166	0.966

References to

Shared 305
Private 1252
S/L(%) 24.

Shared 362
Private 1907
S/L(%) 19.

Table 3A. Problem 3. Features:
Maintaining storage for each generation. At the end there is an edit of the scores. Excluding this part, the eff. would be very close to 1.
** 20 generations.

Table 3B. Problem 3. Features:
Maintaining 5 sets of storage which means we only treat 5 generations at a time. In this case this does not impose a real limitation. Editing is not a critical section.

#PE	Time	eff	References to
1	10842	1.	Shared 478
8	1395	0.97	Private 2797
16	717	0.94	S/L(%) 17.
32	381	0.89	

Table 4. The results of Problem 4. Thirty generations.

Note: The time includes that part spent on the last statistical treatment and print-out. (25 K)

PEs	Hists			
	10^3	10^4	10^5	10^6
32	.9114	.9904	.9990	.9999
64	.8072	.9767	.9776	.9998
512	.2516	.7707	.9711	.9970
1024	.1306	.6002	.9376	.9934
4096	.0300	.2363	.7558	.9687

Table 5. The efficiency predicted by the idealized model

V_m	k/m						
	6	7	8	9	10	12	15
16	.0397	.0146	.0054	.0020	.0007	.0001	.0000
32	.0793	.0292	.0107	.0039	.0015	.0002	.0000
64	.1586	.0584	.0215	.0079	.0029	.0004	.0000
512	1.2691	.4669	.1718	.0632	.0232	.0031	.0002
4096	10.1530	3.7351	1.3741	.5055	.1860	.0252	.0013

Table 6. The expected waiting time in the above case, in units of the time for one history.

(Case)	e^{-t}		p_s^{n-1}	p_c	composite		equal number of summands	
Hists	128	1024	128	1024	128	1024	128	1024
PEs								
4	.968	.996	.971	.996	.972	.996	.821	.936
8	.906	.987	.911	.987	.911	.986	.714	.890
16	.778	.963	.787	.965	.771	.964	.581	.810
32	.587	.915	.601	.919	.556	.919	.423	.700
64	.366	.815	.379	.823	.334	.825	.277	.596
128	-	.652	-	.664	-	.655	-	.461
256	-	.449	-	.463	-	.447	-	.350
512	-	.260	-	.272	-	.254	-	.221

Table 7A. The simulated efficiency for three kinds of distributions and equal numbers of histories.

Note: $P_s = 0.893$, which is derived from the particle transport problem 1 calculated here.

20 experiments were carried out in each case to obtain these estimates.

histories case	100		200		300	
	composite	real	composite	real	composite	real
4	.954	.973	.976	.978	.988	.987
8	.875	.918	.929	.944	.961	.944
16	.715	.789	.824	.842	.898	.858
32	.490	.638	.651	.699	.779*	.734

Table 7B. The comparison between the idealized model and the real simulation

*After the model is improved by taking the batch treatment into account, this value is decreased to 0.738, close to the real value.

p_S	$p_S = 0.50$			$p_S = 0.893$			$p_S = 0.95$		
histories	100	200	300	100	200	300	100	200	300
4	.948	.974	.981	.954	.976	.988	.947	.977	.986
8	.862	.921	.946	.875	.929	.961	.863	.937	.954
16	.709	.820	.875	.715	.824	.898	.724	.857	.881
32	.489	.655	.745	.490	.651	.779	.512	.704	.750

Table 7C. The efficiency for different values of P_S .

Method	Number of PE's	
	Rejection	Direct
1	170015	226007
32	5461	7239

Table 8. The number of steps executed for sampling 1000 points on the unit circle using two methods, with 1 PE or 32 PE's.

Period	2^{46}		2^{62}		2^{70}		
Hist's	n	m=500	m=1000	m=500	m=1000	m=500	m=1000
10^4		.9993	.9986	1.	1.	1.	1.
10^5		.9314	.8675	1.	1.	1.	1.
10^6		.0008	.0	.9999	.9998	1.	1.
10^7		.0	.0	.9892	.9785	1.	.9999
10^8		.0	.0	.3382	.1144	.9958	.9916
10^9		.0	.0	.0	.0	.6547	.4287
10^{10}		.0	.0	.0	.0	.0	.0

Table 9. The nonoverlap probability of the random number sequences in Sec. IV. m = the maximum length of the sequence.

